Storage

Storage requirements

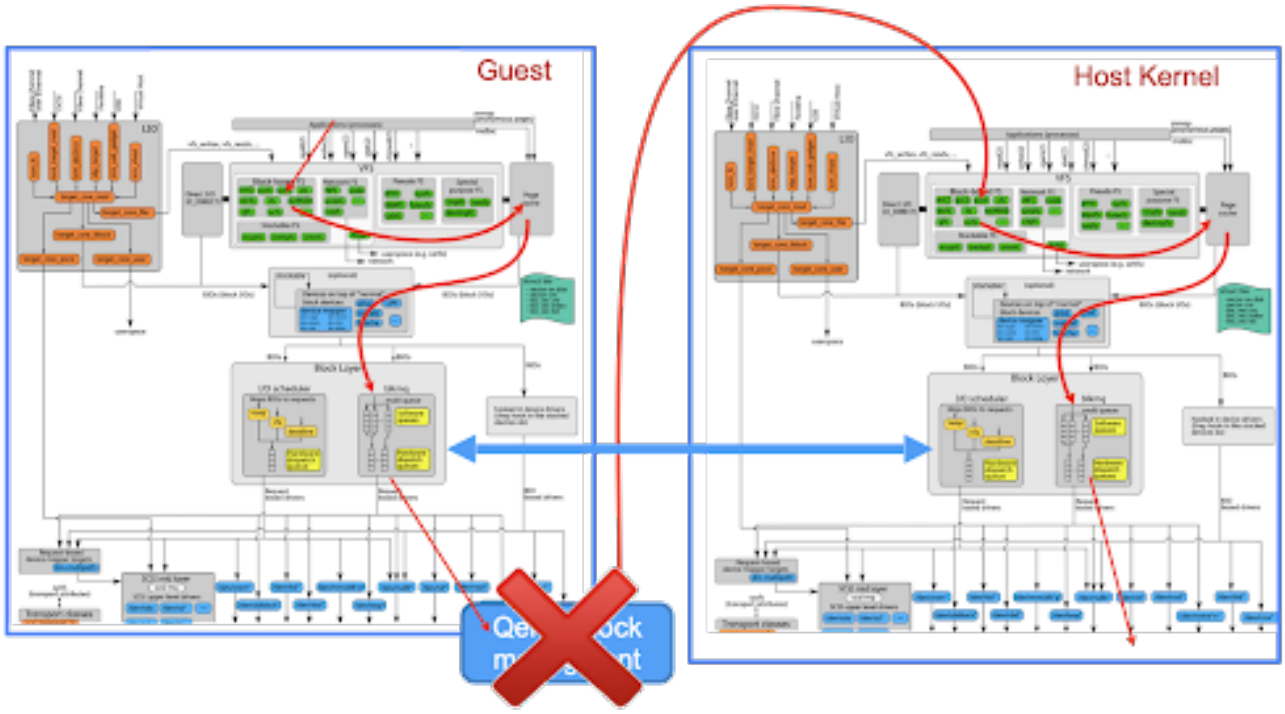
Our requirements from the storage are pretty much the same any other cloud provider has:

- **Full disk encryption**. In more traditional cloud providers this is done to guard users from each other and from cloud provider employees. Guaranteeing that data will never leak. In our conditions we also don't like the idea of the device being stolen, and unencrypted data can be accessed
- **Thin provisioning**. Efficient usage of the storage - the blocks which are not used by a user should not be occupied
- **Snapshotting**. Snapshot a state of guest and easily roll back or forward between the snapshots. Read more about snapshots on Snapshot page.
- **Compression** also adds up to the efficiency of storage usage. And as well might increase read speeds on slower volumes (e.g. emmc)

Problem with current architecture

Poor performance

Our current approach is to rely on qcow2 in order to satisfy mentioned Storage requirements. While it does tick all the boxes, the major problem with it - very inefficient usage of memory and CPU of the host, impossible to parallelize the requests, and notoriously difficult to optimize.



The diagram above shows a typical flow of the disk access of an application running in a virtual machine. The request would have to go through 3 software stacks

- Guest storage stack
- Qemu block management
- Host storage stack

Each stack has its own layer of indirection, and its own cache. Also single disk IO requests cause multiple expensive VM-exits, and user-kernel boundary crosses, which not only takes a significant amount of time but also negatively affects cpu caches. Moreover the architecture serializes all the requests and pipes them down through a single queue.

Lack of compression, snapshotting outside of qcow2

While qcow2 gives us support for compression and thin provisioning, it does it only for virtual machines. But we also serve containers, and this is also a nice to have feature for the eve's own services as well

Lack of redundancy

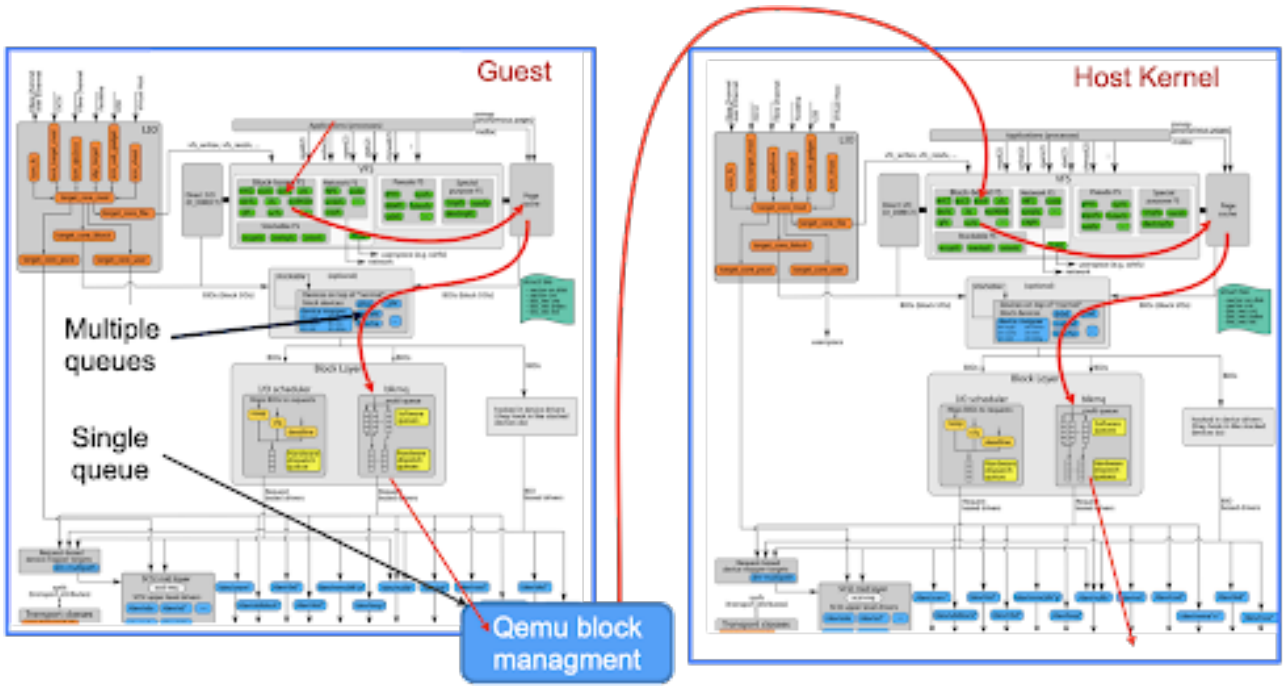
We do not support any sort of raid arrays. Customer cannot join drives into one storage space (potentially more resilient in case of raid with redundancy)

Lack of quotas

We don’t have quotas, it would be nice to limit how much space a service can use without affecting other services, including Eve’s core functionality.

Next generation Storage architecture

So we are taking sort of a default path for a cloud provider. We are getting rid of qemu block management altogether. Ideally we also want to avoid one of the vfs stacks. Preferably the one on the host’s side, because we don’t want to limit customers on what they can run. And that would be awesome if we just create a shortcut from the guest block layer directly to the host’s block layer.



While this is possible by engaging Vhost/Virtio and dedicating a partition on a disk to a virtual machine, we would lose the features mentioned in the Storage requirements chapter (such as thin provisioning). So, we are looking for a compromise.

To address serialization of the requests we are switching to a different protocol which is free from these limitations. Our options were SCSI and NVMe. The latter turned out to be much cleaner and extremely well suited for parallelization – up to 64k IO queues). And, most importantly, starting from version 1.3 NVMe can operate without VMexits, which effectively makes it a paravirtualized protocol, without the need of the special drivers in the guest.

As a transport for the data the natural choice is virtio and vhost which would allow us to deliver data directly to the kernel, to the right place in the kernel. So we can hook up directly into the block layer.

One way to do the thin provisioning would be to direct the data flow to LVM. But a file system with OCI features would give us much richer functionality:

- LVM does support compression and thin provisioning, but the performance penalty is very high, which kills the major benefit of LVM-based solution
- File system has much more context about what is happening with it's blocks: LVM can't know which blocks are free and just keeping junks. Therefore snapshotting and thin provisioning is much more space efficient when implemented in file system layer
- Growing of the disk space takes a lot more steps in LVM (add disk, grow volume group, grow logical volume, grow file system sitting on the virtual media), which in generally can not be done online
- LVM lacks quota support. Once a Logical Volume was allocated to a container, you can't easily change the size of that volume. While in filesystem base approach you would need only change the quota of a dataset
- FS-approach allows to allocate "Project IDs", and count quotas for multiple datasets/files. For example the space allocated for all the containers belonging to Eve could be counted agains one quota

The fs of choice is zfs (unfortunately there is not many file systems satisfying the requirements, currently the list is limited btrfs and bcachefs - the former still is not stable enough especially when it comes to software raid support, the latter is very promising but has to go a long way to become mature).

Intermediate step

While the efforts on NVMe vhost already started, and the first prototype is approaching, we still require time to upstream these efforts, and bring implementation up to production level (extensive testing and performance tuning).

To follow the agile principles we will implement an intermediate solution - Virtio SCSI driver backed by vhost fileio, which, in its turn, is backed by Zvfs Zvol. All the necessary components are already present in the upstream, we would need only to integrate them together in Eve-OS, and execute stress testing.

This approach has additional advantages:

1. Our partner company recently started parallel efforts on improving performance of Zfs Zvol. They would require a playground for their development and benchmarking activities. More importantly, in the bundle Qemu / SCSI_virtio / SCSI_vhost / file_io / Zfs_zvols, zfs_zvols is the least tested component, since it is not used so widely as other features of zfs. Also linux has only recently become an important target for zfs to run on, and may require specific kernel patches to improve stability. But even for the greater final goal we still need to perform this testing, and it is better to start it earlier while we can reach our partner for help.
2. We have important customers which are very interested to have storage with redundancy (aka RAID - Redundant Array of Inexpensive Disks). As of now we do not support software RAID. If zvol stress testing demonstrates level mature enough for production, we can provide this anticipated feature already in September 2021

As current state, Eve already supports Zfs-formatted **`/persist`** and joining multiple disks into redundant arrays.

NVMe Vhost - The greater goal

As of now, we are approaching a first PoC implementation. A fair share of activities so far were investigation, gaining necessary understanding, building support code and environments. Nevertheless, to put it a bit more technical, currently the Guest is able to recognize the device and deliver data to I/O queues, but times out on the response.

The latest problem we had to solve is translation of Guest Physical addresses (which is the responsibility of vmm part of hypervisor - Qemu in our case) to something which Kernel can understand - Host Virtual Addresses. Because with vhost transport Guest communicates directly with the kernel, w/o involvement of Qemu (which is the whole point of Vhost), it might be not trivial to translate those addresses.

We think we have cracked this problem, but unfortunately did not have a chance to prove that the solution is working. Once this is done, we do not anticipate any further big roadblocks, and can enable communication between Guest and Kernel - I/O queues.

People involved in this activity:

- Vitaliy Kuznetsov mostly Qemu side
- Yuri Volchkov - mostly Kernel side

Transition

The biggest challenge here is to deal with the nodes with only one disk attached. New storage architecture requires full reformat, and potentially repartition of that disk. Which is rather hard and error prone to do online. It is very likely that we will end up supporting both storage architectures for a while. This is the least explored problem so far, but below are some options we might provide to the customers who agree to go extra mile for us (keep in mind each option deserves its own document, and we are touching them here only briefly):

- Offer customers to snapshot their applications into the cloud, reboot or kexec into a ramfs-based eve installer, and offline-migrate customer applications back to the Edge-Node
- Ask customers to attach a usb-storage to backup applications and reinstall eve as in last option
- If the node happened to have more then one disk attached, we can provide a button in the zedcontrol to use the spare disk for update process

So far none of the listed options is ideal, and we will explore other possibilities in the future.

Here another advantage of the Intermediate step appears. If a production-ready zfs-based solution comes before the next wave of customers, we will endup with fewer nodes based on the legacy storage format. And transitioning from SCSI-vhost to NVMe-vhost is significantly easier than reformatting the whole system disk.

However we are anticipating another problem related to transition from SCSI-vhost to NVMe-vhost - it is the way how disks are enumerated in the system. For example:

- /dev/vda for virtio-scsi (current Eve implementation)
- /dev/sda for vhost-scsi-pci (implementation described in the chapter Intermediate step)
- /dev/nvme0n1 for nvme emulation

Depending on the way the customer installed their application, they might have a problem. If the rootdevice is specified as /dev/vda, the system would not boot, and manual intervention would be required (it is easy to fix, but it is still extra efforts and pain).

However if the root device is specified as /dev/by-id/<uuid> or /dev/mapper/lvm-<unique-part-id> customer's application will happily boot. Good news is that the second variant is the most used by modern operating systems.

So this transition should happen also only with customer awareness (e.g. customer should explicitly press the button in order to update).

Testing and benchmarking

Fio - Flexible I/O tester is the golden standard for benchmarking storage and file system performance. It is going to be our go-to tool for monitoring performance improvement/degradation.

We will get the first numbers already pretty soon, when the Intermediate step PoC is ready. This would be also interesting to see what is going to be the bottleneck of such architecture, and potentially adjust the development plan according to the discoveries.

At the current (or the soonest future) state of the project, it would be interesting to collect the following benchmarking numbers:

- Current storage architecture
- Native disk performance
- SCSI_vhost with zvol
- SCSI_vhost with lvm
- Upstream implementation of nvme emulation in qemu

As for stress testing, the preliminary plan is the following:

- Days of running fio and fsstress
- Host sudden power off under the heavy I/O load (e.g. fio)
- Swarm of guests (e.g. 16) are generating heavy load simultaneously, while creating snapshot periodically

Performance tuning

Storage stack we are working on has multiple components, and a lot of surface for performance tuning. Starting from reducing the number of VM-exits and finishing with CPU cache coherency. We are going to rely heavily on the efforts Ruslan Dautov is busy with - special debug build with Vtune included. This tool is a nice addition to the perf which is already available in Eve-OS. Vtune provides great insights on what CPU is busy with, including mentioned VM-exits and cache problems.

Integrating efforts on OpenZFS's zvol performance improvements

Our partner is working on zfs tuning and code optimisation with the focus on our specific workload (using Zvols at the edge for backing customer storage). We are going to incorporate the results of there investigations continuously. However all the changes to zfs shall be upstreamed unless not feasible to upstream. That means that some of this work will be already included into the first prototype described in a chapter Intermediate Step.

Storage is fairly critical component, thus we need at least initial testing before rolling out the updated architecture to the customers. Therefore as the first step of integrating we have decided to invest some efforts into tests, as described in the chapter Testing and benchmarking.

Milestones / Next steps

- End of August 2021: initial storage stress tests
- End of September 2021
    - `/persist` partition formatted as ZFS, transport vhost-scsi or emulated NVMe in the guest
    - Prototype implementation of NVMe-Vhost emulation
- End of October 2021: Submission of the first version of the patches implementing NVMe-Vhost in Linux and Qemu; Integrated ZFS into protype
- End of November 2021: Upstreamed patches; Performance tuning, bug fixes
- End of December 2021: Extensive testing; First production ready version w/o transitioning existing instances to new storage format (not implemented/tested yet)
- Never ending: Continuous performance optimisation and bug fixing

References

1. Project repository
2. Snapshots in EVE